GRE®

RESEARCH

Validation and Extension of the Mathematical Expression Response Type: Applications of Schema Theory to Automatic Scoring and Item Generation in Mathematics

Mark Kevin Singley and Randy Elliot Bennett

July 1998

GRE Board Professional Report No. 93-24P

ETS Research Report 97-19

Educational Testing Service, Princeton, New Jersey 08541

Validation and Extension of the Mathematical Expression Response Type: Applications of Schema Theory to Automatic Scoring and Item Generation in Mathematics

Mark Kevin Singley and Randy Elliot Bennett

GRE Board Report No. 93-24P

July 1998

This report presents the findings of a research project funded by and carried out under the auspices of the Graduate Record Examinations Board.

Educational Testing Service, Princeton, NJ 08541

****

Researchers are encouraged to express freely their professional judgment. Therefore, points of view or opinions stated in Graduate Record Examinations Board Reports do not necessarily represent official Graduate Record Examinations Board position or policy.

****

The Graduate Record Examinations Board and Educational Testing Service are dedicated to the principle of equal opportunity, and their programs, services, and employment policies are guided by that principle.

Acknowledgements

This work was funded by the Graduate Record Examination Board; we gratefully acknowledge their support. We also thank our many colleagues at Educational Testing Service (ETS) who contributed greatly to this work: Dave Bostain and Ken Berger of the Information Systems and Technology (IS&T) Division at ETS programmed the production version of the Mathematical Expression scoring algorithm, and Alex Vasilev, also of IS&T, programmed the production interface. Dan Jacquemin, of the ETS Research Division, performed many of the analyses of the accuracy of the scoring algorithms. The Schema Prover was implemented in large part by Krishna Jha and Pei Wang, ETS Research Division consultants. The Schema Compiler was implemented in large part by Eric Gold, also an ETS consultant. And finally, we thank Mary Morley, of the ETS Research Division, who has acted as our mathematical advisor and has contributed countless ideas and suggestions regarding many aspects of this work.

Abstract

In this report, we describe the development of general, accurate, cost-effective, and immediately usable automatic analysis routines that dichotomously score rational expressions of arbitrary complexity. This development has led to the new Mathematical Expression (ME) response type, which may appear operationally in the new GRE Mathematical Reasoning test scheduled for deployment in 1999. We detail work confirming the accuracy of these scoring routines and then describe extensions to the existing capability based on schema theory. We show how schema theory can be applied not only to the scoring of more complex test responses but also to the automatic generation of items and intelligent tutoring.

The first generation of computer-based testing has provided important advantages, especially to examinees who now have more opportunities to test, more comfortable examination environments, and faster score reporting. All the same, this first generation has not altered the substance of testing in any fundamental sense; we measure the same constructs using items that are essentially the same as before. Future generations of tests will allow us to present questions that expand what and how we measure (Bennett, 1994). One means for expanding what and how we measure is by changing the nature of the response from a forced choice to a machine-scorable construction. In mathematics, we have been developing and evaluating approaches to machine-scorable constructed responses since 1988. Using a program called GIDE (Sebrechts, 1992), this research centered upon automatically analyzing the solution process, that is, the sequence of steps the examinee used to arrive at a final result. These process analyses resulted in a partial-credit score and a diagnostic characterization of the solution. Our research has found the partial-credit scores to agree highly with those awarded by human content experts and with performance on the GRE General Test's quantitative section (Bennett, Sebrechts, & Rock, 1991; Sebrechts, Bennett, & Rock, 1991). Similarly, the diagnostic characterizations, which described errors detected in the solution, agreed with the consensus of human judgments in most cases (Bennett & Sebrechts, 1996). However, the "knowledge-based" analytic approach we used involved labor-intensive preparation for each narrowly defined class of problems. As a consequence, that approach was better suited to instructional and test preparation applications than to high-stakes admissions testing, where new items need to be continuously created.

One important outcome of this research was the realization that, for certain problem classes, more efficient approaches to automatic analysis could be applied, especially if error diagnosis was not a requirement (as would generally be true for admissions tests). In particular, Sebrechts, Bennett, and Katz (1993) recommended adapting the technology used in such commercial symbol manipulators as Theorist, Mathematica, and Maple to score solutions that take the form of algebraic expressions. We worked with the GRE Mathematical Reasoning (MR) Test Team at Educational Testing Service (ETS) to prototype and then create a production-ready version of a response type that meets this goal. The resulting capability works with any item whose answer is a rational expression. In this report, we describe work confirming the accuracy of the underlying algorithm and then describe extensions to the existing scoring capability based on schema theory. In particular, we describe extensions that involve the scoring of more complex test responses and the automatic generation of items, both of which were goals of this project. As a side benefit, we show how this theory makes possible the delivery of sophisticated instruction that may lead to important improvements in test preparation products.

I. The Mathematical Expression Response Type

In its prototype form, the scoring algorithm underlying the Mathematical Expression response type was based on public domain source code that performs algebraic

symbol manipulation developed in the computer science department at the University of California at Berkeley. From this LISP source code, we developed general, accurate, cost-effective, and immediately usable routines that dichotomously scored rational expressions of arbitrary complexity. The routines were general in that they could be used for *any* test item whose answer was a rational expression. They were accurate in that they appeared to work 100% of the time. They were cost-effective in that entering the key for a new item took seconds, not the days, weeks, or even months required in knowledge-based systems. Finally, the routines were immediately usable, as evidenced by the fact that they were employed to score responses to computer-delivered Mathematical Expression items included in the November 1993 GRE Mathematical Reasoning pilot test. Following the pilot test, the Mathematics Automatic Scoring Initiative contributed to developing a production-ready version of the response type. This version included an examinee interface, tutorial, test developer tool, and scoring routine. Figure 1 illustrates the examinee interface. The examinee uses the mouse and a "soft keyboard" to enter arbitrarily complex rational expressions. The interface also has a built-in syntax checker that notifies examinees when their responses are malformed (e.g., unbalanced parentheses, illegal juxtaposition of operators).

For efficiency's sake, the scoring routine was completely redesigned; in its production implementation, it bears little, if any, resemblance to the original prototype. As noted, the scoring capability can dichotomously process rational expressions; that is, multivariate polynomials that may or may not involve division. Such expressions include:

$b/2$

$$\frac{(x^3+y^2)z}{x+y}$$

$$\frac{(m-2p)*(n-2p)}{4}$$

$x^2+.7x+.12$

The problem in recognizing the correctness of expressions such as these lies in the fact that there are an infinite number of ways to express the same mathematical relationship. This is the problem of *mathematical paraphrase*. For example, the third expression above has the following equivalent forms (and infinitely many more):

$$\frac{(n-2p)*(m-2p)}{4}$$

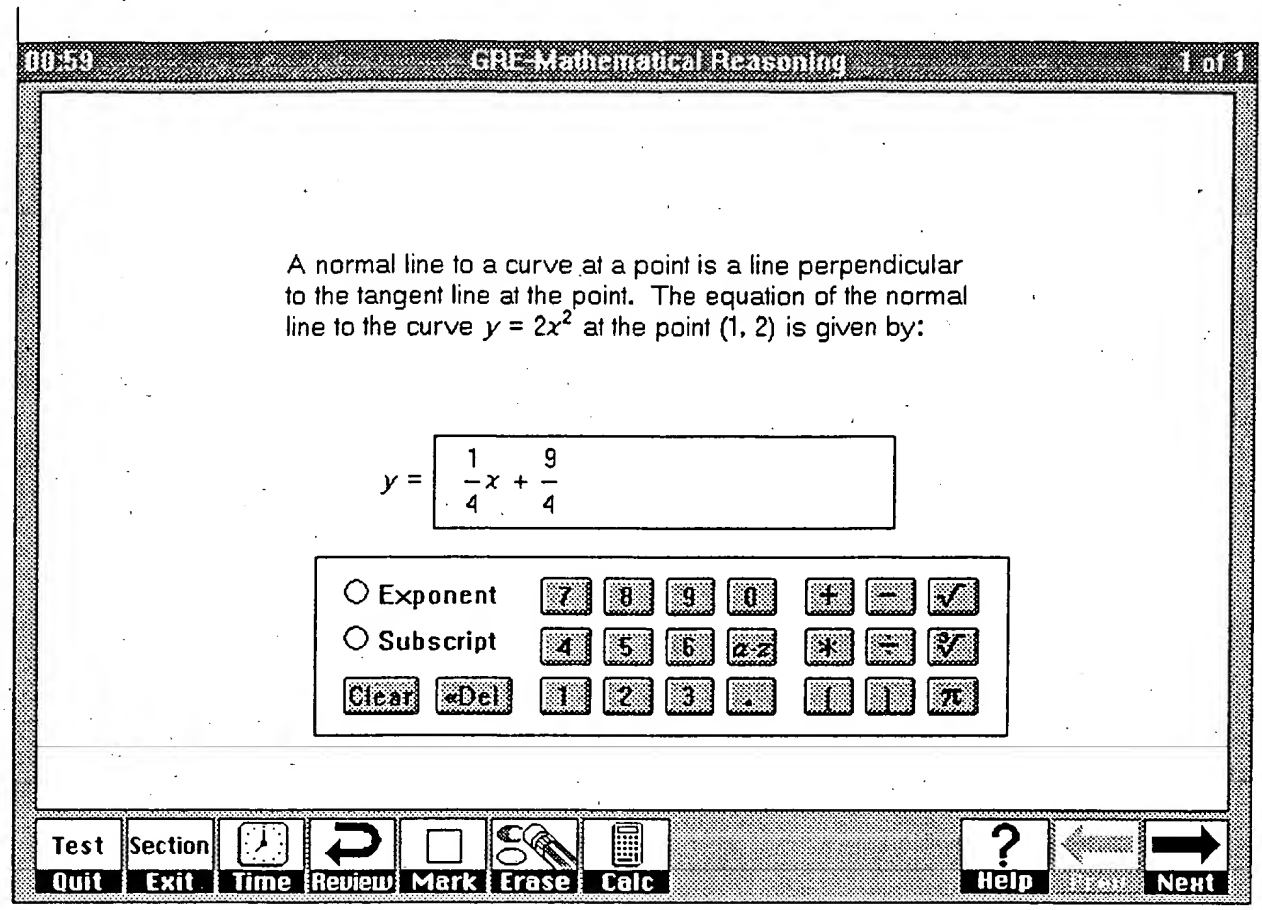


Figure 1. The examinee interface for the Mathematical Expression response type showing an example item with an incorrect response.

$$.25(-2p+m)*(-2p+n)$$

$$p^2 - pn/2 - pm/2 + mn/4$$

The scoring program uses principles of symbolic computation to reduce equivalent rational expressions to a single normal form. (This normal form is far removed from the surface-level representation of these expressions.) Once in this form, expressions can be easily matched for algebraic equivalence with the key.

The subfield of symbolic computation within computer science has made enormous strides in the last two decades. Twenty-five years ago, the simplification and canonicalization of rational expressions was not well understood and was considered to be an advanced topic in artificial intelligence. Today, sophisticated algorithms have been specified and ongoing work in the field is devoted to fine-tuning these techniques (Norvig, 1993).

Validation of the Single-Line Mathematical Expression Scoring Capability

We evaluated the accuracy of both the prototype and the production-ready implementations of the ME scoring capability. The prototype was evaluated using data collected from the November 1993 MR field trial. These data were gathered in computer-based form using an early version of the interface and consist of 185 examinees' responses to each of four items. Each response was scored for the pilot test by a mathematics test developer. For our purposes, the responses were rescored by the algorithm and disagreements reviewed by a mathematics test developer, one of the authors, and a technical assistant. Figure 2 gives the four items and their keys.

Results are given in Table 1. Of the 654 nonblank responses in the data set, 226 (35%) were deemed to be correct literals or paraphrases of the key by our review. As the table shows, the algorithm and the judges agreed on the disposition of all but 12 of the 654 responses. Resolution showed five of these disagreements to be errors on the part of the human grader. In the remaining seven cases, the expressions contained fractional exponents and, thus, were beyond the recognized capability of the algorithm. Unlike the interface used in the pilot test, the current version disallows entry of such irrational expressions, requiring the examinee to simplify before submitting them for scoring.

Because it had to be redesigned to run under the speed and memory constraints characteristic of low-end test-center equipment, the production-ready algorithm has additional limitations beyond the requirement for rational expressions. When these limits are exceeded, the algorithm returns an "indeterminate" score that indicates the general reason for the rejection. The limitations are the following:

Given $f(x) = e^{x^2 - x}$,
find a second degree polynomial $p(x)$ such that

$$p(0) = f(0)$$
$$p'(0) = f'(0)$$
$$p''(0) = f''(0)$$

$p(x) =$ 1-x+3/2x^2

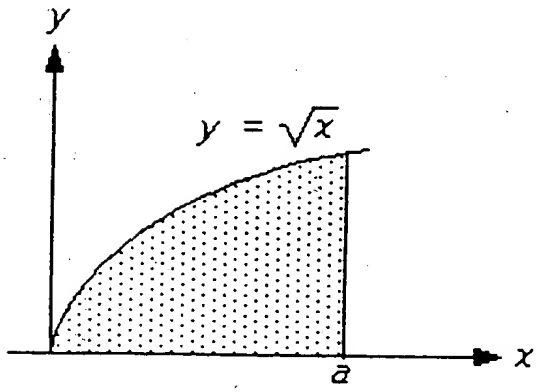


Figure I

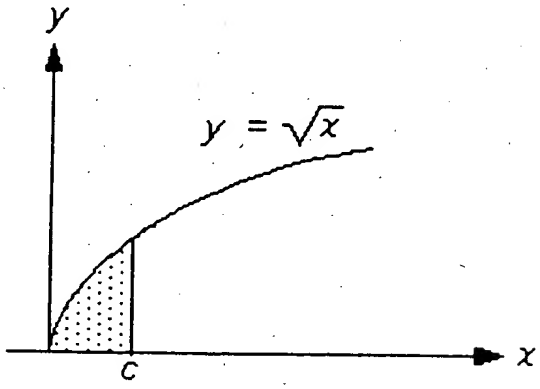


Figure II

The area of the shaded region in Figure I is 8 times the area of the shaded region in Figure II.

What is c in terms of a?

$c =$ (1/4)a

Figure 2. The four Mathematical Expression items used in the MR field trial.

$$x + by = d$$
$$cx + y = e$$

b, c, d, and e are nonzero constants.

If the system of equations has infinitely many solutions, what is c in terms of b ?

$c =$ 1/b

A normal line to a curve at a point is a line perpendicular to the tangent line at the point.

The equation of the normal line to the curve $y = 2x^2$ at the point $(1, 2)$ is given by

$y =$ (-1/4)x+(9/4)

Figure 2, continued.

Table 1
Disagreements Between Prototype Scoring Algorithm and Human Judges (n=185)

Item	Number of Nonblank Responses	Number of Right Responses	Observed Disagreements
1	169	57	1
2	165	53	0
3	145	40	3
4	175	76	8
Total	654	226	12

Note: The number of right responses is the number of answers to the problem judged as correct by our review.

- No intermediate result in the processing of an expression can have more than 256 terms. This restriction is owed to the limited memory space available after the ETS computer-based testing software, the ME interface, and the computer operating system are loaded.

- Exponents can only be integers or single variables; they cannot be fractions or polynomials. If an expression is raised to a variable power, the expression is treated as if it were being raised to the first power. For example:

  $(a + b)^x = (a + b)^1$

- Expressions can be raised to an integer power depending upon the number of terms being raised:

Table 2
Limits on the Use of Powers

Number of Terms Being Raised	Maximum Exponent
1	300
2	32
3	12
4	6
5	4
6	3
7	3
8	2
9	2

- Polynomials may be raised to a negative power, but the maximum value to which they can be raised is considerably lower than the same polynomial raised to a positive value.

- Small numbers may lose precision because all calculations are integer based. Each decimal is treated as a fraction composed of two integers (e.g., 0.25 is converted to

25/100). Significant digits are removed to avoid exceeding the limitation on large terms.

- Scoring of an expression must be completed within a fixed time. This limit was set to prevent the algorithm from introducing too much "wait" time into the adaptive testing process.

The production-ready algorithm was evaluated in three stages. First, we used the same set of 654 pilot test responses that were run through the prototype algorithm. As Table 3 shows, 53 of the 654 nonblank responses contained complex exponents (either fractional or polynomial) and, thus, the production algorithm could not render an accurate right/wrong judgment. (The prototype algorithm, which can use more memory, was able to handle polynomial exponents.) In all cases, the current production interface would not have allowed these complex exponents to have been entered. Putting aside these aberrant entries, the production algorithm disagreed with the human judges in only 4 of the remaining 601 instances, and in all four the production algorithm proved correct.

In the second stage of our evaluation, we used data from 1,320 examinees participating in the October 1995 and January 1996 ME field trials. Twenty ME items were spiraled among seven test forms in the October trial and an additional 13 items were spiraled among four forms in the January trial.

We submitted the item responses from these trials to the algorithm and then checked the results manually. Of the 6,834 nonblank item responses, 26 were disagreements, each of which the algorithm marked as wrong and we would have scored as right. The 26 diagreements fell into two categories (see Table 4). Twenty-three of the

Table 3
Disagreements Between Production Scoring Algorithm and Human Judges for Pilot Test Responses (n=185)

Item	Number of Nonblank Responses	Number of Right Responses	Number with Complex Exponents	Observed Disagreements
1	169	57	0	1
2	165	53	0	0
3	145	40	39	3
4	175	76	14	0
Total	654	226	53	4

Note: The number of right responses is the number of answers to the problem judged as correct by our review.

Table 4

Results of Running Production Algorithm on Field Trial Responses (n=1,320)

Item	Number of Nonblank Responses	Number of Right Responses	Number of Responses with Subscripts	Number of Otherwise Right Responses with Wrong Subscripts	Number of Otherwise Right Responses with Superfluous Text
			October 1995 Field Trial		
1	219	150	0	0	0
2	256	207	0	0	1
3	369	62	0	0	0
4	228	152	1	0	0
5	457	322	0	0	0
6	244	165	0	0	0
7	228	47	181	3	0
8	227	102	0	0	0
9	217	158	0	0	0
10	358	203	320	10	0
11	236	123	0	0	0
12	169	114	0	0	0
13	483	190	1	0	0
14	242	189	0	0	1
15	234	201	0	0	0
16	135	39	0	0	0
17	151	70	0	0	0
18	211	102	195	10	0
19	241	129	0	0	0
20	219	164	0	0	0
			January 1996 Field Trial		
1	132	75	0	0	0
2	122	101	1	0	0
3	135	106	0	0	0
4	115	33	0	0	0
5	136	64	0	0	0
6	145	121	0	0	0
7	146	127	0	0	1
8	136	57	4	0	0
9	97	44	2	0	0
10	144	124	0	0	0
11	152	71	1	0	0
12	140	74	0	0	0
13	110	80	0	0	0
Total	6834	3964	706	23	3

Note: The number of right responses is the number of answers to the problem judged as correct by our review.

disagreements involved examinees using subscripts incorrectly. Subscripts were required in the responses for 3 of the 33 items and 706 of the item responses used subscripts. (Forty-eight of these uses were incorrect and in 23 of them the response was marked wrong solely for that reason.) Two kinds of confusion prevailed. In one problem where the answer was to be stated in terms of r_0, several candidates entered r_o (i.e., with the subscript being the letter "o" instead of zero). This problem might be addressed by calling attention to this potential confusion in the ME tutorial, changing the font to distinguish the characters better, or even by removing the letter "o" from the variables and constants menu in the ME interface altogether. Alternatively, the scoring algorithm might be relaxed to treat "o" and "0" equivalently when they appear as subscripts. A second confusion was that some examinees failed to denote their subscripts as such, for example, entering CI instead of C_I, again causing the algorithm to misinterpret the response. This confusion might also be handled through the tutorial or by relaxing the scoring algorithm to treat these two forms as equivalent.

The second type of erroneous entry involved examinees using the variable and constants menu to enter text strings. Of 6,834 item responses, 3 otherwise correct responses included unit abbreviations (e.g., "mi" for miles), which the scoring engine treated as concatenated variables and thus misinterpreted. (Ten other responses also included text strings but were incorrect for unrelated substantive reasons.) This difficulty was anticipated in that the current ME tutorial instructs examinees to use letters only to represent variables and constants. Also, concern for the entry of superfluous text was one of the reasons behind the decision to disable the computer keyboard and require the examinee to build responses using the mouse only. Again, greater emphasis might be placed in the tutorial on the need to avoid text entries. Additional edit checking also might be added to the interface to query the examinee whenever common abbreviations are detected.

In the third stage of our evaluation, we asked GRE mathematics test developers to construct a set of 20 difficult, yet plausible, key expressions and their paraphrases, so that we could assess the limits of the algorithm in detecting algebraic equivalence. The 20 key expressions were particularly designed to test the grading of multivariate equations, multiple nesting of operations, and round-off error. The expressions are shown in Table 5.

Before running the test developers' paraphrases through the algorithm, we constructed a smaller test set of our own. These paraphrases revealed several problems with the algorithm, which were corrected prior to running the developers' paraphrases as cross-validation.

Table 6 gives the results of the cross-validation. Two hundred ninety-two difficult paraphrases were generated by the developers. Because of the complexity of the expressions, 12 of these paraphrases turned out not to be algebraic equivalents of the keys but are still included in the analysis. The algorithm was able to process 205 (70%) of the paraphrases, all of which it scored accurately. The algorithm rejected the remaining 87 paraphrases (returning scores of "indeterminate"), because the expressions exceeded one

Table 5
Twenty Difficult Key Expressions Used to Test Production Algorithm

1.	$\left\{\frac{(1000*(x-1)^2+1)}{1000}\right\}^4$	11.	$\frac{x}{y}+\frac{y}{z}+1$
2.	$\frac{(1+2x+x^2+x^3)}{(1+2x+3x^2+x^3+x^4)}$	12.	$\frac{a}{(b+c)}+\frac{b}{(a+d)}$
3.	$\frac{(x^2+2xy+y^2)}{(2x^2+7xy+3y^2)}$	13.	$\frac{(ed-bf)}{(ab-cd)}$
4.	$\frac{(2x^2+y^2+3xy+x+y)}{(2xz+yz+z+2x+y+1)}$	14.	$\frac{1}{(\frac{1}{a}+\frac{1}{b}+\frac{1}{c})}$
5.	$\frac{(a+b)(b+c)}{(a-b)(b-c)}$	15.	$\frac{(x-a)^2}{r^2}+\frac{(y-b)^2}{s^2}$
6.	$\frac{(a-1)(b-1)(c-1)}{abc-1}$	16.	$((x+1)^2+\frac{3}{x})^2$
7.	$\frac{(a^2+2ab+b^2+16)}{(a^2+6ab+9b^2+1)}$	17.	$(\frac{x}{y}+1)^2(y+z)$
8.	$\frac{(a+b)^4}{((a+b)^4+1)}$	18.	$(\frac{1}{(x+y)})(\frac{1}{x}+\frac{2}{y})$
9.	$(a+1)(b+2)(c+1)(d+2)$	19.	$\frac{(1+ab+ac+bc)}{abc}$
10.	$(x+y)^2+(y+2z)^2+(x+2z)^2$	20.	$(((x+y)^2+y)^2+y)^2$

or more of its limitations (e.g., too many terms, too long to process). Seventy-six of these 87 rejected paraphrases were associated with five key expressions (1, 4, 8, 16, and 20), for which the algorithm rejected all paraphrases. Because all 87 rejects were syntactically correct, they *would* have been allowed by the current interface. Rejects can be avoided in an operational test by writing items that require relatively simple expressions. Also, the complexity of a response can be limited operationally by adjusting the size of the response window in the ME interface. This adjustment can be made on an item-by-item basis and

Table 6

Results of Running Difficult Expressions Through Production Algorithm

Key Expression	# of True Paraphrases	# of False Paraphrases	# of Paraphrases Rejected
1	17	0	17
2	12	0	2
3	16	0	0
4	6	6	8
5	15	0	2
6	11	0	1
7	13	0	1
8	20	0	20
9	14	0	1
10	11	0	1
11	16	0	0
12	10	0	1
13	11	0	1
14	18	0	0
15	15	0	0
16	15	1	15
17	10	5	0
18	17	0	1
19	17	0	0
20	16	0	16
Total	280	12	87

Note: "True" paraphrases are ones that were algebraically equivalent to the key. "False" paraphrases were not algebraically equivalent to the key. The number of true and false paraphrases equals the total number of paraphrases.

is under the control of the test developer. However, if an examinee does pose an overly complex response for some reason, the algorithm is able to quickly eject it and return a message asking that the examinee simplify and resubmit the entry.

II. Extensions: Partial Credit, Instructional Applications, and Item Authoring

We believe that the Mathematical Expression response type is both a useful current capability and a foundation upon which to build. In its current form, ME represents an initial foray into performance assessment in mathematics supported by automatic scoring. However, ME does not provide for partial-credit scoring or the analysis of multiple-line responses.

Being able to track and score multiple-line responses is important from two perspectives. First, it provides a basis upon which to assign partial credit. Presumably, if an examinee spends a significant amount of time solving a complex problem, one would like to make finer discriminations than simply whether the answer is completely right or wrong. The test may be throwing away information that took valuable time to collect if a complex response is only summarized as a single "bit." Second, it offers information

about the individual steps or the path the examinee took toward solution. Whereas information about precisely how the examinee solves problems may not strictly be needed for admissions testing, it should be valuable for self-assessment, particularly when supplemented with error and other qualitative process information.

When attempting to develop automatic scoring routines for multiple-line responses, one necessarily learns a lot about the underlying structure of the problems. This knowledge of problem structure is indispensable to making progress in the analysis, but it may also be useful for other applications. In particular, it may be useful for either offering remediation in an instructional application or for automatically generating items. In the remainder of this paper, we report on our attempts to apply a particular theory of problem structure, schema theory, to the analysis of multiple-line solutions, the delivery of instruction, and the automatic generation and variation of items.

Schema Theory as a Basis for Automatic Analysis

In mathematics, many problems involve multiple steps that lead to a final solution. When examinees work on paper, many of these intermediate steps are written down, leaving a trace of the solution process that could be subjected to analysis. We have been working to develop algorithms that can analyze such multi-step responses in terms of correctness and completeness.

We have restricted our attention to problems that can be construed as involving linear systems of equations. Many standard algebra word problems fall into this category. For example, here is an algebra word problem that appeared on a recent test in multiple-choice format:

> Excluding rest stops, it took Juanita a total of 10 hours to hike from the base of a mountain to the top and back down again by the same path. If while hiking she averaged 2 kilometers per hour going up and 3 kilometers per hour coming down, how many kilometers was it from the base to the top of the mountain?

The key to our analysis of responses to such problems is the assertion that word problems can be characterized (and categorized) in terms of the underlying set of equations that relate the entities of the problem to one another. According to this analysis, problems that superficially appear quite distinct may in fact be instances of the same underlying problem structure, or *schema* (Mayer, 1981). The Juanita problem is an instance of the *round-trip* schema. The round-trip schema involves the following equations:

(1) $d_t = d_u + d_d$
(2) $t_t = t_u + t_d$
(3) $d_u = r_u * t_u$
(4) $d_d = r_d * t_d$
(5) $d_t = r_t * t_t$
(6) $d_u = d_d$

This set of variables and primitive equations defines an entire class of problems. Given a set of variables and equations such as this, a particular problem within the class is represented as a set of variable assignments and a goal. Table 7 defines the schema variables and gives their values in the context of the Juanita problem. Table 8 organizes the schema equations into a table. One of the relationships in the table, $r_u + r_d = r_t$, does not hold. However, the other relationships are correct.

Table 7
Variables, Their Meanings, and Their Values.

Variable	Meaning	Value
d_u	distance up	x (goal)
d_d	distance down	unknown
d_t	distance total	unknown
t_u	time up	unknown
t_d	time down	unknown
t_t	time total	10 h
r_u	rate up	2 km/h
r_d	rate down	3 km/h
r_t	rate total	unknown

Table 8
Round-trip Schema Equation Table.

	part	+	part	=	whole
distance =	d_u		d_d		d_t
rate *	r_u		r_d		r_t
time	t_u		t_d		t_t

An interesting property of schemas is that, for any particular problem instantiation, not all of the underlying primitive equations may be required for solution. This is what gives the schemas psychological content and differentiates them from purely structural descriptions of problems. In a purely structural description, only the equations required for solution are included. However, in a schematic description, the entire set of relationships that model the situation are included. In essence, the schema position states that certain sets of relationships tend to co-occur in the world and therefore tend to cohere in a person's head. When someone is confronted with such a situation (e.g., when someone is asked to solve an algebra word problem), the entire set of relationships is activated and brought to bear to try to understand the situation. One large part of the problem is determining which subset of the equations contained in the schema is actually relevant for solving the current problem (Singley, 1995).

For example, in the sample problem, equations (2), (3), (4), and (6) are required, but equations (1) and (5) are not. However, this is impossible to determine a priori. To solve the problem, the relevant equations first must be identified, then instantiated with the given information, and finally composed together to create an equation that contains only the goal variable:

$$\frac{d}{2}+\frac{d}{3}=10 \qquad \text{where } d = d_u \text{ or } d_d$$

Using algebra, one can determine from this equation that $d = 12$, and the problem is solved.

In summary, the schematic account of algebra word problem solving involves the following steps:

- activation (generation or retrieval) of the appropriate schema based on comprehension of the problem statement
- identification of the relevant equations from that schema for the current problem
- instantiation of the relevant equations with problem information
- composition of relevant equations into a solvable equation
- algebraic solution

We do not wish to imply that examinees actually go through these steps in the exact order or granularity shown above. Particular problem-solving strategies may combine and/or interleave the above steps in a variety of ways.

In the remainder of the paper, we will report our efforts to date to apply this basic analysis to (1) support analysis of multiple-line responses and partial-credit scoring, (2) build instructional applications that can provide immediate feedback and remediation for emerging problem solutions, (3) automatically and/or semi-automatically generate items, and (4) make predictions concerning item difficulty.

Analysis of Multiple-Line Responses

We have shown above a sample solution for the Juanita problem, an instance of what we call the round-trip schema. Unfortunately from the standpoint of analysis, there are many other ways than the one shown for an examinee to arrive at an answer to this problem. Furthermore, there are many ways in which an examinee might evidence some

level of partial understanding in an errorful solution. For example, an examinee might show the following work when solving the Juanita problem:

$$t_u + t_d = t_t$$
$$t_t = 10$$
$$2d + 3d = 10$$
$$d = 2$$

This solution is incorrect, but much of the work shown is valid and it therefore may be deserving of partial credit. How do we systematically analyze such responses?

Of course, the problems of mathematical paraphrase and numerical imprecision that needed to be addressed in the analysis of ME responses still hold in the analysis of multiple-line responses. But in addition to these problems, there are many others.

- Many different strategies for solving a problem exist; all valid strategies must be accommodated in the analysis.
- Substeps may be combined; the examinee is free to determine the granularity of each step.
- Substeps may be skipped; certain calculations or symbolic transformations might be done in the examinee's head.
- Substeps may be inconsistent; one step may directly contradict another.
- Substeps may be partially correct; a single step may be composed of correct and incorrect elements.
- The solution trace may make use of multiple symbol systems (e.g. figures, tables, mathematical symbols, natural language).

For the purposes of the present work, we have assumed that the examinee's response will be composed entirely of syntactically correct expressions or equations involving numbers and mathematical symbols, and all the variables that appear in the solution must be defined in terms of their problem referents. We have chosen to defer the analysis of supporting figures and tables for now.

The Schema Prover

Given these assumptions about the kind of input we receive from examinees, we apply techniques from *constraint logic programming* to analyze the examinee's response. Constraint logic programming is an extension to logic programming (e.g., Prolog) that allows for the declarative representation of a quantitative problem as a set of linear constraints. Once represented in this way, the answer to the problem can be easily *proven*

by propagating the constraints. If the problem is underdetermined (i.e., there are an insufficient number of constraints to uniquely determine the solution), the system can derive the most constrained functional relationship between the goal variable and other variables in the problem. We are using constraint logic programming, not to solve problems directly, but rather to determine the relationship between a canonical representation of the constraints of the problem (the schematic description) and an examinee's response. For the current project, we built a system called the Schema Prover that attempts to derive the examinee's response from the schematic description as a kind of *proof*.

As outlined above, we first develop a structural, schematic description of the problem we are analyzing. This description is composed of a set of primitive equations, a set of given variables, and a goal variable. With this problem description in hand, we do a two-pass analysis of the response.

Correctness: We first determine whether the response is correct, that is, whether what is presented in the response is consistent with the problem description. To do this, we try to determine whether the response can be derived from the problem description. We frame the problem in terms of a logical proof: Given the problem description, is it possible to *prove* the response? If it is, we at least know that the equations and values of variables presented in the response are consistent with the problem description. However, we still do not know whether the examinee has represented *all* of the problem constraints in the response and has derived the final answer.

If we determine that the examinee's response is *not* correct, we then systematically vary the constraints in the problem description and try again to prove the correctness of the response. In other words, we try alternative sets of premises (alternative problem descriptions) and see whether any of these sets can prove the conclusion (the response). For example, we may propose a different problem description that replaces all primitive constraints of the type $d = rt$ (equations 3, 4, and 5) with the errorful variant $t = rd$. If, after degrading the problem description in this way, we are successful in proving the response, we have a very specific account of the error in the response: We know which constraint has been misrepresented and in what way. (It turns out that in the sample response shown above, two errors of this type are present in the third line.)

Completeness: Once the analysis for correctness is finished, we must analyze the response for completeness. To determine whether the response is complete, we simply reverse the premises and conclusions of our earlier proof: Given the examinee's response, is it possible to *prove* the problem description? If it is, then we know that at some level all of the relevant constraints are represented in the response, and the response is complete. If it is not, we attempt to determine what is missing in the response. We do this through a procedure very similar to the one outlined above for correctness: We systematically supplement the response with

constraints from the problem description and attempt to determine the minimal set of constraints from the problem description that, when added to the response, make it possible to solve the problem. In this way, we can once again pinpoint precisely which constraints are missing from the examinee's response. For example, consider the following correct yet incomplete response to the Juanita problem:

$$d_u = 2t_u$$
$$d_d = 3t_d$$
$$2t_u = 3t_d$$

All of the work shown is correct, but the examinee has failed to provide all of the constraints necessary to uniquely determine the value of d_u or d_d. By systematically supplementing this response with constraints drawn from the problem description, we can determine that, with the addition of the constraints $t_t = t_u + t_d$ (equation 2 above) and $t_t = 10$, the problem is solvable. Thus, once again we have a precise description of what is present and what is missing in the response.

In sum, our approach involves representing a particular problem as a set of constraints, and then determining which (if any) of those constraints is violated and/or absent in the response. By reducing a particular response to the set of quantitative constraints it represents, the analysis bypasses many of the problems outlined earlier concerning variations in the surface forms of responses: multiple strategies, skipped steps, composed steps, and partially correct steps. According to this approach, two responses are equivalent if they can be reduced to the same set of primitive constraints.

Such an approach lends itself well to the implementation of partial-credit scoring rubrics. Subscores can be associated with each constraint in the problem description. The presence or absence of the constraint in the response can either increment a base score or decrement a total score for the problem, respectively. Also, subscores can be associated with common errors, i.e. common degradations of the problem description. In addition, such rubrics need not be created for each individual problem; they can be created once for an entire class of problems if they are done at the level of the problem schema.

Instructional Applications of Schema Theory

The constraint logic programming system described above for analyzing multiple-line responses is optimized for a certain kind of scoring task: the assignment of a partial-credit score on the basis of an analysis of an entire response. This optimization resulted in an analytical engine that is less than optimal for instructional applications, however, for the following reasons:

(1) The constraint-logic analysis is holistic in that responses are characterized rather abstractly in terms of the presence or absence of correct and buggy schema equations in the entire response. Analyses of individual lines of the response are not performed, and the internal structure or logical progression of the response is not appreciated. Thus, the system is not in a very strong position to offer remediation.

(2) The analysis is optimized to detect completely correct responses or near-misses: responses that deviate from correct responses by the relaxation or absence of one or two constraints. This was done because presumably any response deserving of partial credit would necessarily be fairly close to the correct response. Thus, the system devotes most of its attention to doing a very thorough exploration of the space of responses very close to the correct response, and relatively little attention to those further away from the correct response. As a result, the system does less well analyzing partial responses (or responses-in-progress), which an instructional system presumably would be called upon to do.

We envision an instructional system based on schema theory that would not simply assign a score to a student's multiple-line response, but would offer immediate feedback and remediation on each line of the response as it is being entered. Its analytical capabilities would strive to approach those of a human tutor: the system would understand exactly where the student was in the solution of the problem, and would be able to offer help to the student based on a dynamic analysis of the best path from the student's current state to the goal state of the problem (Singley, Anderson, & Gevins, 1991). In some cases, it would also be able to detect and remediate errors. (Such a system might be thought of as a future addition to the GRE PowerPrep series.) In order to build such a system, we have designed and implemented as part of the current project another schema-based analytical engine (a companion to the constraint-logic system described above) that is optimized for instructional applications.

The Schema Compiler

The goal of the Schema Compiler is to take a minimal schematic description of a math problem (a set of schema equations, a set of variable assignments, and a goal variable) and generate from it a solution graph that represents all possible problem states and all valid transitions between states. An instructional system should be able to use such a graph to (1) represent whatever state a particular student is in and (2) generate the "best" path from the student's current state to the nearest goal state. The Schema Compiler generates a solution graph in two basic steps:

> *Step 1: Determine means-ends solutions.* First, the system must determine whether the problem is solvable, and if so, which subset (or distinct subsets) of schema equations are required. It does this not only for the goal variable of the problem, but for every unknown. (In the instructional system, a student should not be limited to determining just the stated goal of the problem, so paths involving the derivation of other unknowns need to be modeled.)

The system uses a general problem-solving strategy borrowed from artificial intelligence and cognitive psychology known as *means-ends analysis* to determine these sets of equations. Means-ends analysis is a depth-first, recursive, working-backward strategy that has been widely used as a method for solving systems of equations (Newell & Simon, 1972; Larkin, McDermott, Simon, & Simon, 1980). In means-ends analysis, the program first retrieves a schema equation that contains the goal variable. (If more than one schema equation contains the goal variable, there is more than one starting point for this process; this is represented as an "OR" node in the search tree.) This equation would be examined to determine whether or not the goal variable is already constrained sufficiently so that its value is determined. (For example, if all the variables in this first equation other than the goal variable are known, the process terminates: only a single equation is required to solve for the goal.) However, if there are other unknowns in the first equation, subgoals are set recursively to find values for them. In order for this ultimately to become an equation solvable for the goal variable, the values of *all* the other unknowns must be determined, so these subgoals are linked by an "AND" node in the search tree. As the system recurses, additional schema equations are retrieved that contain these other unknowns, and again the system checks to see whether the variables are sufficiently constrained to allow for solution. This process is performed exhaustively to generate a search tree that represents all possible means-ends linkages of equations. A particular branch of the search tree terminates when either (1) the system runs out of schema equations to retrieve and apply (failure condition) or (2) the schema equations retrieved serve to constrain the goal variable sufficiently to allow for solution (success condition).

Figure 3 shows the means-ends analysis search tree generated while solving for the goal variable (d_u) of our example Juanita problem. (Again, similar search trees are generated for the other unknown variables of the problem.) As shown in the figure, there are six distinct solution branches generated, but only two of these lead to success. This reinforces the claim made earlier that, in addition to activating the appropriate schema, a significant part of problem solving in this domain is determining which subset of schema equations is required to solve the current problem.

Step 2: Derive problem graph. Once all the means-ends solutions have been found (in the Juanita problem, there are eight: two each for each of the four unknown variables), the Schema Compiler uses them to generate the problem graph. The problem graph represents all possible problem states that can be derived from the schema and all valid transitions between states.

In our example, the means-ends analysis for the goal variable d_u produced two solutions. In both cases, these solutions amounted to linear chains of

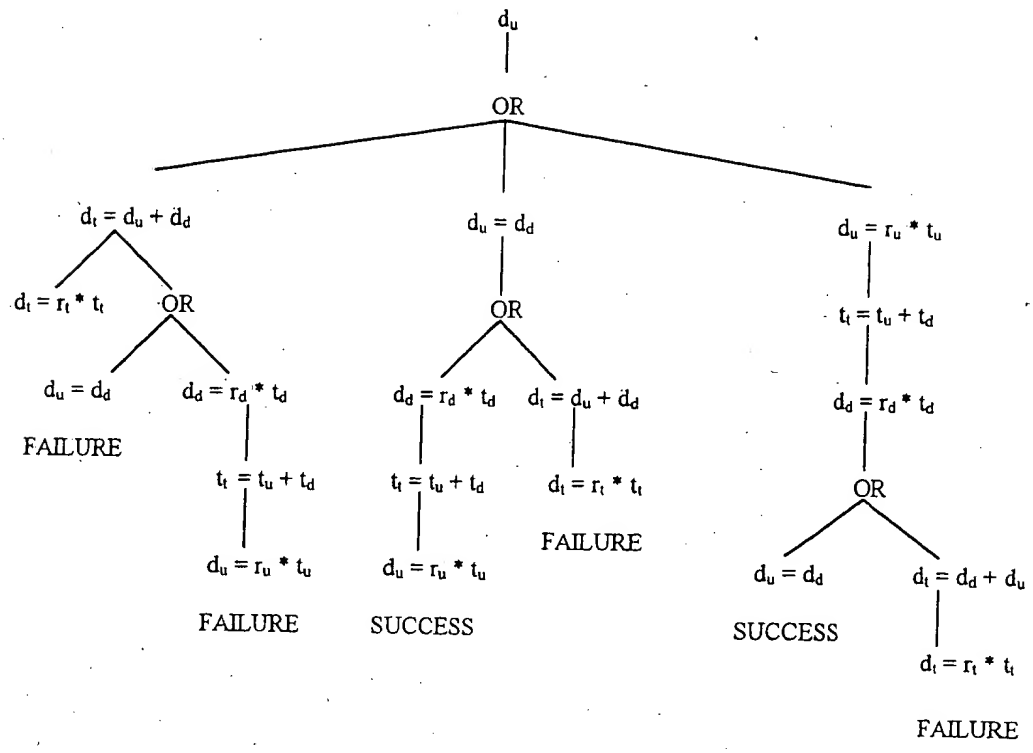


Figure 3. And-or search tree generated by means-ends analysis for the goal variable (d_u) of the Juanita problem. Any node not marked as an "OR" node is an "AND" node. Terminal nodes of each branch are marked with either "success" or "failure."

equations.[1] The task of the Schema Compiler at this point is to generate all permutations and combinations of the schema equations on these paths, while preserving the linkages in the original search tree.

Figure 4 presents the portion of the problem graph generated by the Schema Compiler for the two means-ends solutions for d_u. (This is a small fraction of the entire graph, but perhaps that portion most frequently traversed by students who are making progress solving the problem.) The rectangular nodes of the graph represent all the different equations a student might write that are directly on-path to solving for d_u, and the links between the nodes are all the possible ways these nodes might combine to produce more complex equations. The circular nodes that join the links represent the variables that are replaced when one node is composed with another. One rectangular node in the graph (2,3,4,6) is distinguished in that it represents the equation that can be solved for the value of d_u. In this case, it represents the particular composition of four primitive schema equations that eliminates the unknown variables d_d, t_u, and t_d but preserves the goal variable d_u. (In the full graph, there are other distinguished nodes that can be solved for the values of the other unknowns.)

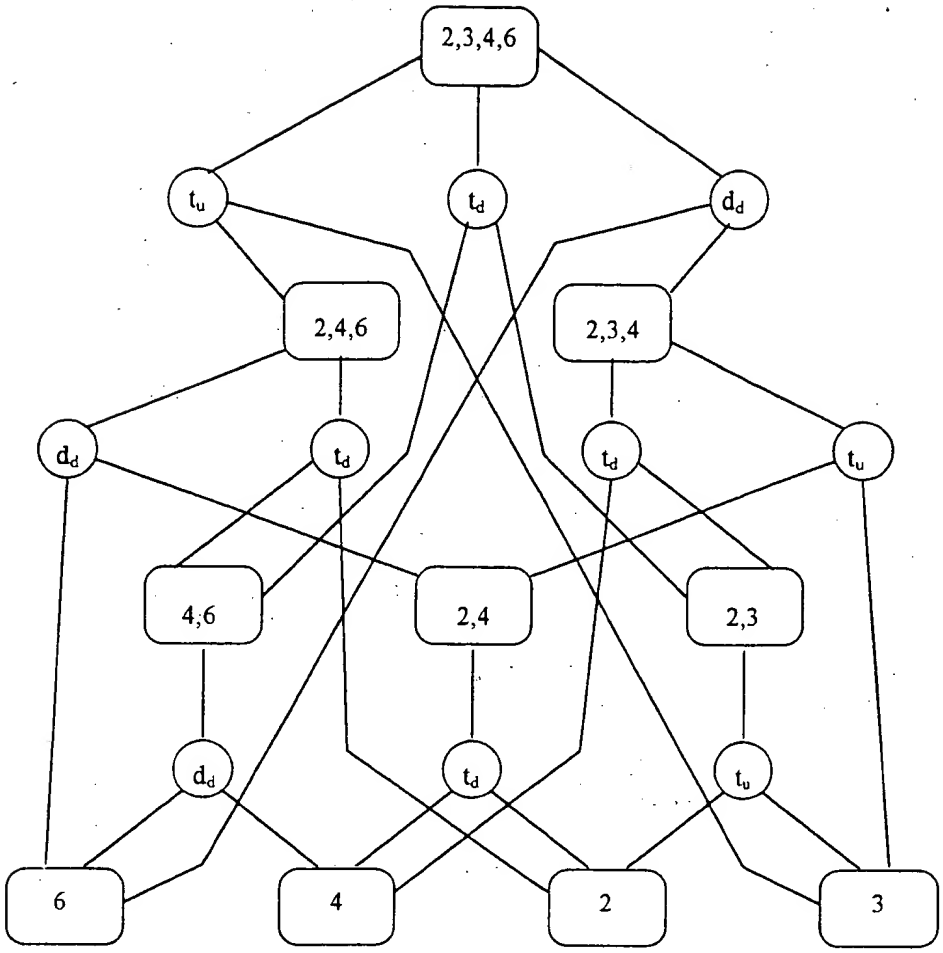
This, then, is how the problem graph is constructed. However, several complications should be mentioned:

Each rectangular node in the graph does not represent a single equation but, in fact, an entire set of equations. This is because students can write the same equation at different levels of instantiation (i.e., they can either use values for known quantities or variables). So, for example, the equation $d_u/2 + d_u/3 = 10$ could alternatively be written by the student as $d_u/r_u + d_u/r_d = t_t$. This equation is perfectly valid; it simply does not make use of all the information given in the problem. In this example, because there are three known quantities in the equation, there are 2^3 different ways the equation might be instantiated.[2] All the different equations that can be written for a single node are called the node's *projections*. The one projection that contains values for all known quantities in the problem is called the node's *modal projection*.

The logic of the problem graph is really based on linkages between the modal projections of the nodes. If the student has not written equations that are modal projections, those equations must be turned into modal projections before the logic of the problem graph applies. This is the problem of *underinstantiation*, and it is fairly trivial to address: Presumably, the instructional system would encourage the student to plug in values for any variables used by the student that are given values in the problem statement. A different, thornier problem arises when, as an intermediate step in solving the problem,

[1] The solutions were linear chains because, in each schema equation retrieved and applied, there was a single unknown. This rendered all of the AND nodes in the successful portions of the graph degenerate. Other topologies (e.g., trees) for solutions are possible, however.

[2] We are currently dealing only with equations that are under-instantiated, i.e. equations where variables are being used for known values.



Nodes:

(2) $10 = t_u + t_d$
(3) $d_u = 2 * t_u$
(4) $d_d = 3 * t_d$
(6) $d_u = d_d$

(2,4) $10 = t_u + (d_d/3)$
(2,3) $10 = (d_u/2) + t_d$
(4,6) $d_u = 3 * t_d$

(2,4,6) $10 = t_u + (d_u/3)$
(2,3,4) $10 = (d_u/2) + (d_d/3)$
(2,3,4,6) $10 = (d_u/2) + (d_u/3)$

Figure 4. A small portion of the problem graph for the Juanita problem. Shown are the nodes and links for the modal means-ends analysis solutions for the goal variable, d_u.

the student solves for the value of a variable that is not the goal variable of the problem. This is the problem of *overinstantiation*: a variable other than the goal variable changes its status from unknown to known in the middle of the problem. Because the logic of the means-ends analysis is entirely predicated on the designation of certain variables as either known or unknown, this invalidates the logic of the problem graph. To deal with this problem, we in fact generate means-ends solutions not only for the problem as it is originally stated, but also for the problem restated to include values for all the unknown quantities that (a) are replaced in the modal means-ends solutions and (b) whose values can be derived from the original problem. Thus, the means-ends solutions we have described so far are in fact only the *modal* means-ends solutions. We run the analysis a total of 2^5 (32) additional times to determine the best solutions when the values of other unknowns have been derived midstream in solving the problem.[3]

Finally, it should be noted that, in addition to the correct schema equations, we have allowed for "buggy" versions of those equations to be submitted to the Schema Compiler. Thus, the Schema Compiler can generate subgraphs that include equations representing common errors, as well as compositions of errors with correct equations or compound errors. (The inclusion of compound errors in the graph might in many cases result in a combinatorial explosion that would be prohibitive, however.)

To summarize our example, when we submit the Juanita problem to the Schema Compiler with one additional buggy equation ($d = r/t$ rather than $d = r * t$), the resulting problem graph consists of 32 nodes having 142 distinct projections and 64 distinct linkages. The total number of distinct problem states that can be represented by the graph is therefore 2^{142} different combinations of projections times 2^5 different levels of over-instantiation for a total of 2^{147} different states. Although the number of possible states for these problems is astronomical, the graph that represents these states is actually fairly compact. The graphs can be precompiled and stored efficiently so that it is practical to use them in real-time instructional applications, as illustrated below.

An Application of the Schema Compiler: The Algebra Word Problem Tutor

The problem graph generated by the Schema Compiler can be used to understand in precise detail exactly where a student is in solving a schema-based problem and to deliver advice about how best to complete the solution given the student's current state. With College Board sponsorship, we have been working on a prototype instructional system, the Algebra Word Problem Tutor, that has at its core the problem graphs generated by the Schema Compiler. Thus, the prototype uses some of the basic analytic infrastructure generated in the current GRE project.

[3] The number of additional means-ends analysis is 2^5 because there are five unknowns in the problem other than the goal variable. Each of these may or may not be determined midstream in the problem.

The system is designed to collect multiple-line solutions from students as they solve schema-based algebra word problems. As students write equations with the tutor, the system matches those equations to individual projections in the problem graph using an evaluation-based matching strategy. The system can detect buggy equations written by the student and offer appropriate remediation. When the student asks for help, the system inspects the student's current state and, using the logic of the problem graph, *dynamically* computes the best path between the student's current state and the nearest goal state.

Figure 5 shows the tutor interface. Students can write any number of equations to model the problem using the soft keypad shown. (This is a slight modification of the soft keypad used for the GRE MR Mathematical Expression response type.) As equations are written and recognized as correct by the system, they are displayed in chronological order in the "Equations So Far" window. (The figure shows the screen of the tutor as the student is writing the third equation.) As equations are written, students are asked to bind all variables they use by selecting from a list of noun phrases drawn from the problem statement. These variable bindings are displayed in a separate "Variables" window.

In addition to providing for the entry of multiple equations, the interface provides the student with a number of powerful symbolic computation operators to manipulate those equations:

- **Substitute Equations:** Allows students to plug one equation into another and thereby reduce the number of unknown variables.

- **Isolate Variable:** Allows students to rewrite any equation so a particular variable is by itself on either the left- or right-hand side. This is often a prerequisite to the Substitute Equations operator above.

- **Solve Equation:** Provides the solution to an equation if the equation has a single unknown.

By providing these symbolic operators, we allow students to focus primarily on the higher level representational and conceptual aspects of solving word problems and de-emphasizing the more rote, procedural aspects. Providing these operators also serves to reduce the amount of time students have to spend interacting with the soft keypad, thus reducing total response time as well as the chance of low-level errors or slips.

Tools for Automatic Item Generation

Aside from automatic scoring and instructional applications, we have also devoted some attention in the current project to issues of item authoring, including automatic or semiautomatic generation methods. Once again, the cornerstone of our thinking is the notion of a problem schema, a set of variables and constraints that defines a problem's

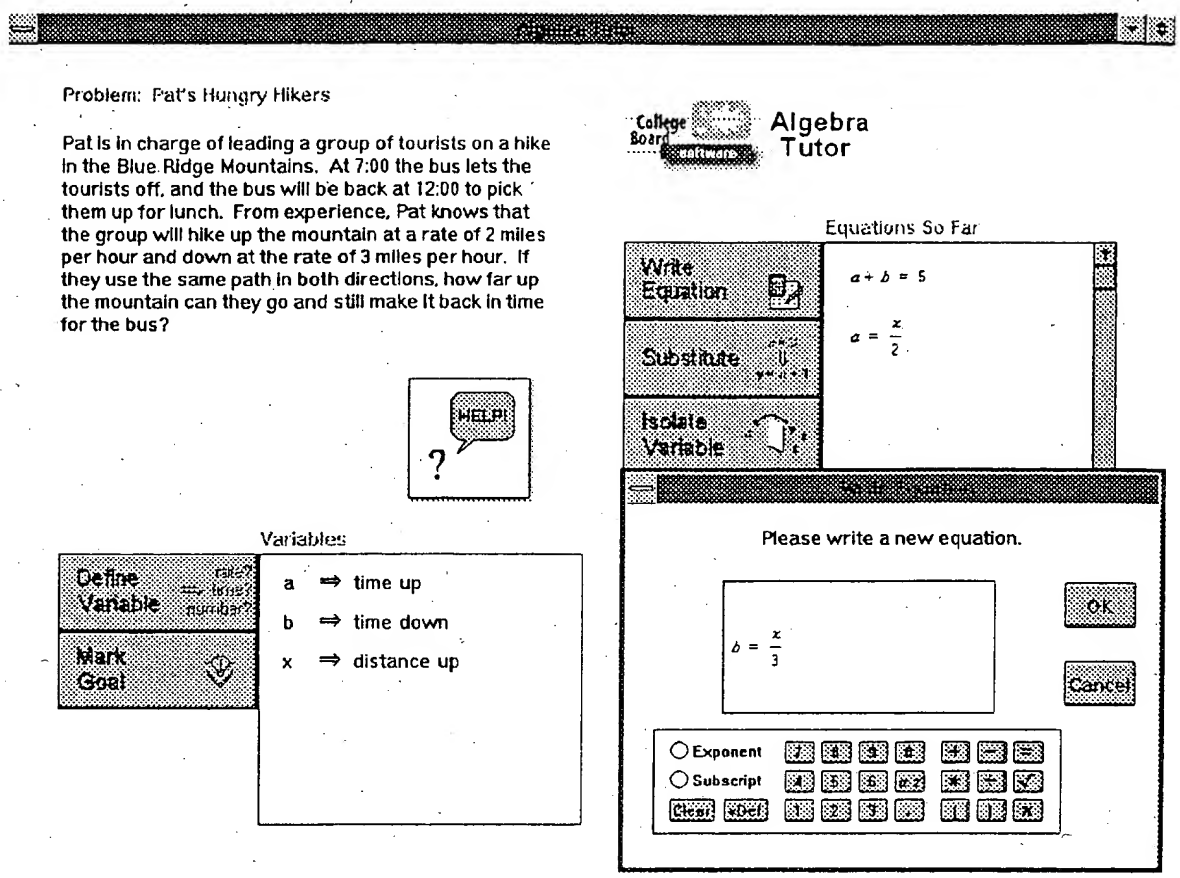


Figure 5. The interface of the Algebra Word Problem Tutor.

deep structure. Given this notion of a schema, a wide spectrum of item generation possibilities exists, ranging from the deep to the superficial:

- Given a set of primitive equations (e.g., distance = rate * time, part + part = whole), generate a set of interesting problem schemas. These schemas could be arranged into a taxonomy of sorts, and could serve as a repository of ideas for test developers.

- Given a particular schema (e.g., the round-trip schema defined above), generate interesting structural elaborations. This involves adding one or more constraints. For example, in the round-trip schema, one could add some relationship between the rate for the first leg and the rate for the second; e.g., the rate uphill is half the rate downhill.

- Given a particular schema, generate all the possible problem structures. A problem structure is defined as a particular configuration of given and goal variables.

- Given a problem structure, generate and/or select a problem context. Thus, the round-trip schema could be instantiated in a hiking context, a boating context, and so forth.

- Given a context, vary the noun referents for problem variables and generate different sets of values for given variables. For example, "Juanita" becomes "Tom" and the total time changes from 10 to 20 hours.

- Given a context, noun referents and variable values, generate a natural language cover story.

We envision embodying the above functionality in an integrated system that offers more to the test developer than just item generation capabilities. With supplemental funding from the ETS Program Research Planning Council, we have been developing a vision of a Test Developer's Assistant that, in addition to automatic and semiautomatic item generation, would offer two other distinct types of functionality:

> *1. Item creation, classification, and tracking.* These are basic word processing and database functions that form the backbone of functionality for the system. Users should be able to type in items, edit them, import graphics, and so forth. Items that are created with the system should be compatible with test delivery software. Item management features should allow the test developer to track the location and status of an item throughout its life cycle. In addition, items would be arranged in a searchable and browsable library in terms of whatever conceptual frameworks are used in the automatic generation and/or analysis of items. The text/graphics of these library items should be directly accessible by the item creation tools; that is, the user should be able to edit the text of a library item to create a new item.

2. *Item analysis.* The system should be able to perform analyses of items that attempt to predict level of difficulty in a principled way. The analyses will be based on the availability of psychological theory that relates features of the items to item statistics (e.g., Sheehan & Mislevy, 1994; Tatsuoka, Birenbaum, Lewis, & Sheehan, 1993). Depending upon the set of features that need to be extracted, the analysis may proceed completely automatically or may require some human assistance. To the extent that well-defined cognitive structures can form the basis of item generation, fairly deep automatic analyses should be possible.

The Algebra Word Problem Test Developer's Assistant

Using the schema theory formulated in the present project, we have built an item creation tool, called the Algebra Word Problem Test Developer's Assistant, that is our initial attempt at fleshing out the vision described above. Figure 6 shows the main system view, which consists of two primary components:

The *math item library* is a repository of math items hierarchically organized by their underlying schematic structures. The hierarchy is defined in terms of shared schema equations, such that a child schema inherits all the equations of its parent schema, and perhaps supplies one or two more. Parent-child relationships between schemas are designated in the interface by indentation in the list of schemas, as is done in many displays of directory-subdirectory structures for computer file systems.

In the figure, the round-trip schema is selected in the Math Item Library. Shown below the selection in the Schema Equations window is the set of six equations that define this schema. Notice that the round-trip schema is a child of the two-legged DRT schema, which in turn is a child of the simple (or one-legged) DRT schema. In each of these cases, more equations are added as one moves down the hierarchy. The simple DRT schema has only the first equation shown for the round-trip schema, and the two-legged DRT schema has the first five but not the sixth.

Presently, all entries in the schema hierarchy are created by hand with a set of authoring tools that are part of the Math Item Library. Nineteen schemas are currently defined, but of course this is a very preliminary set. The library of schemas needs to be stocked further under the direction of test development staff. In the near future, we would also like to provide the system with the ability to suggest extensions to the library itself, either by proposing completely new schemas or by elaborating existing schemas. In this way, the hierarchy could in essence "grow" itself. Of course, test developers should be given the opportunity to accept, reject, or modify these system-generated extensions.

Once an item is selected in the Math Item Library, its contents appear in a structured *Item Editor*. The editor we provide currently supports standard five-option multiple-choice items, but other editors could be easily created for other item types. The editor allows the user to modify the stem and/or any of the options, and also to designate

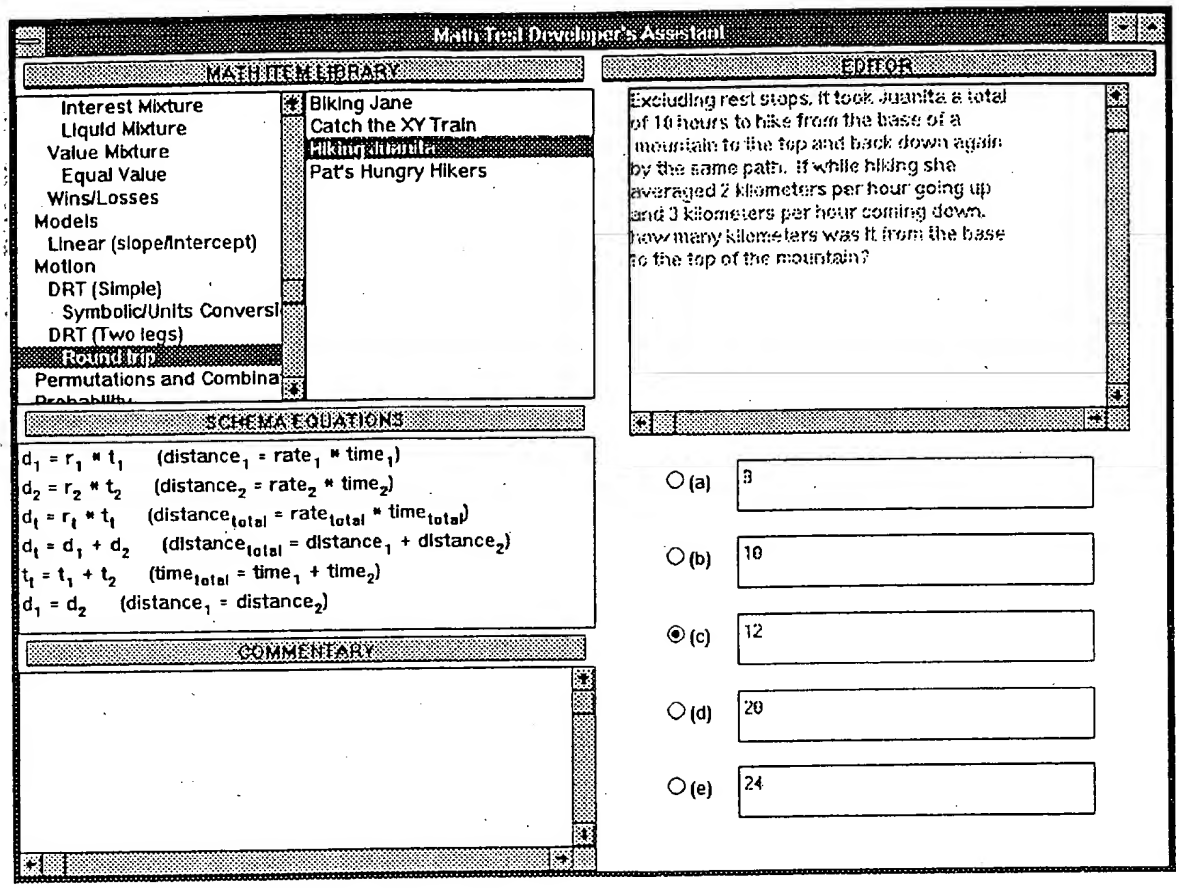


Figure 6. The interface of the Math Test Developer's Assistant. The location of the Juanita problem in the schema hierarchy is shown.

the key and its position among the options. After modification, the user can redeposit the item back in the library using its old name, or can create a new item with a new name. Thus, the system supports the creation of new items from old items (i.e., cloning) by hand.

In addition to tools for the hand-cloning of items, the system provides a tool for more automatic cloning that makes direct use of the problem schemas. Figure 7 shows the Math Item Incubator tool opened on one of the items in the library, the Lawyer's Invoice problem. This problem is an instance of the Linear Model schema, which is characterized by the single equation $y = mx + b$. Using the Math Item Incubator, the user can transform this item into an item template that can be used to generate variants.

Creating an item template is basically a two-step process: First, the user replaces literal strings of text and/or numbers in the original stem and options with variables. Second, the user defines constraints that tell the system something about how the variables should be instantiated. A critical feature of the system is that these constraints are defined declaratively: the user simply states the constraints and it is the system's responsibility to figure out how to simultaneously solve them.[4]

When the Math Item Incubator is opened on an item, the system automatically defines as constraints the equations that characterize the item's schema. The variables in the schema equations automatically become variables that can be referenced in the template, and any values supplied by the system for these variables will satisfy the schema equations.

As an example, Figure 8 shows one possible template for the Lawyer's Invoice problem. Any element in the stem or options prefixed by the syntactic marker '@' denotes a variable element in the template. All remaining elements are literal elements that remain constant across variants. As shown in the template, some of the variables require non-numeric values (e.g., @person) and others require numeric values (e.g. @y). In the former case, the user may either type in a list of values or browse a database of math contexts built specifically to support the instantiation of non-numeric variables. The Math Context Browser currently consists of hierarchically-organized categories of common persons, places, and things that might appear in math problems. Figure 9 shows the Math Context Browser with the category Professional Services (a subclass of Occupations, which in turn is a subclass of People) selected. By simply making this selection, the user can specify how a non-numeric variable should be instantiated.

In the case of instantiating numeric variables, again the user can elect to simply type in a list of alternatives. Another possibility is to define a range for the variable, as well as a way to step through that range (e.g., the variable may be specified as ranging from 50 to 150 in steps of 5). By specifying the magnitudes as well as step sizes, the user can control critical features of the variants that may factor into determinations of

[4] Alternatively, the user could have been required to write a procedural program that would explicitly derive the values of all variables and would propogate the constraints between variables. This we felt would have rendered the system unusable by nonprogrammers.

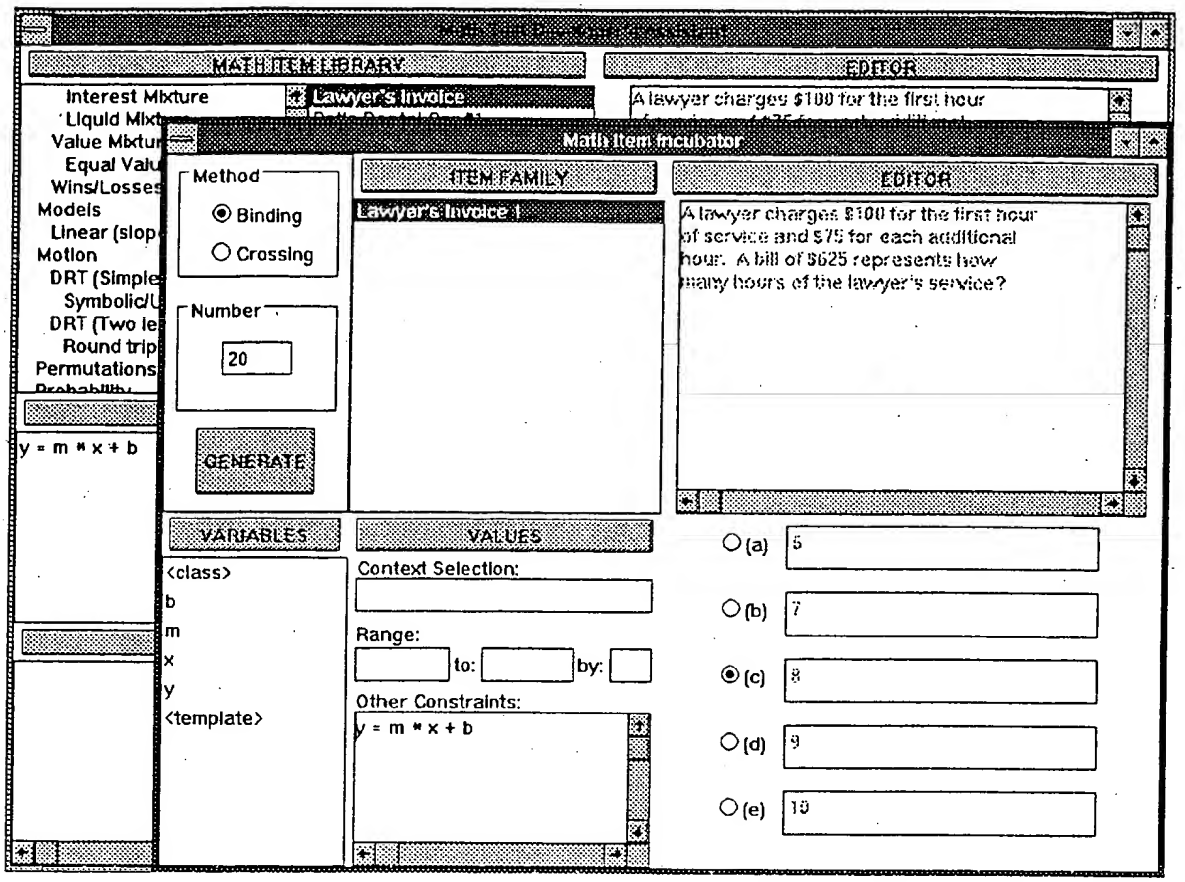


Figure 7. The interface of the Math Item Incubator. The Lawyer's Invoice problem is about to be turned into a template.

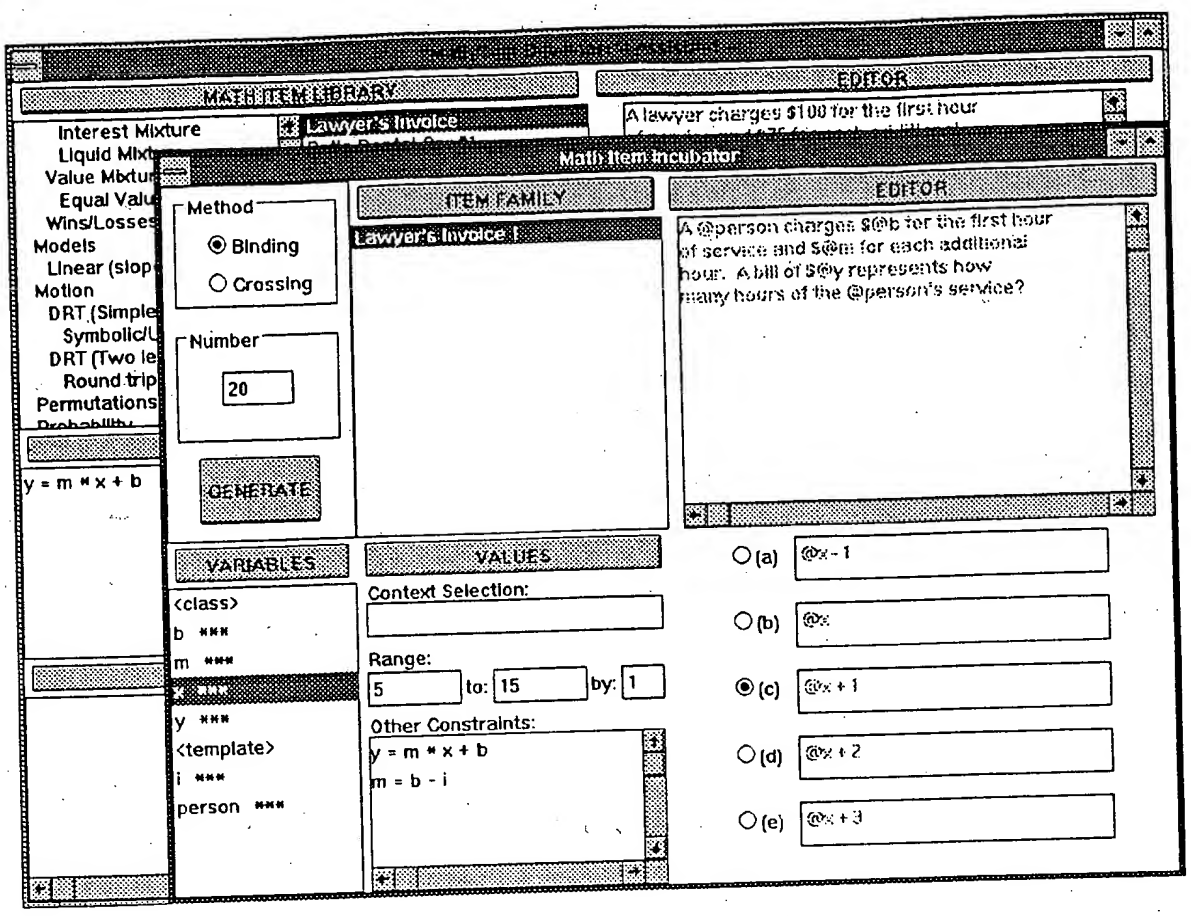


Figure 8. One possible template for the Lawyer's Invoice problem. Variables are prefixed by the syntactic marker '@'.

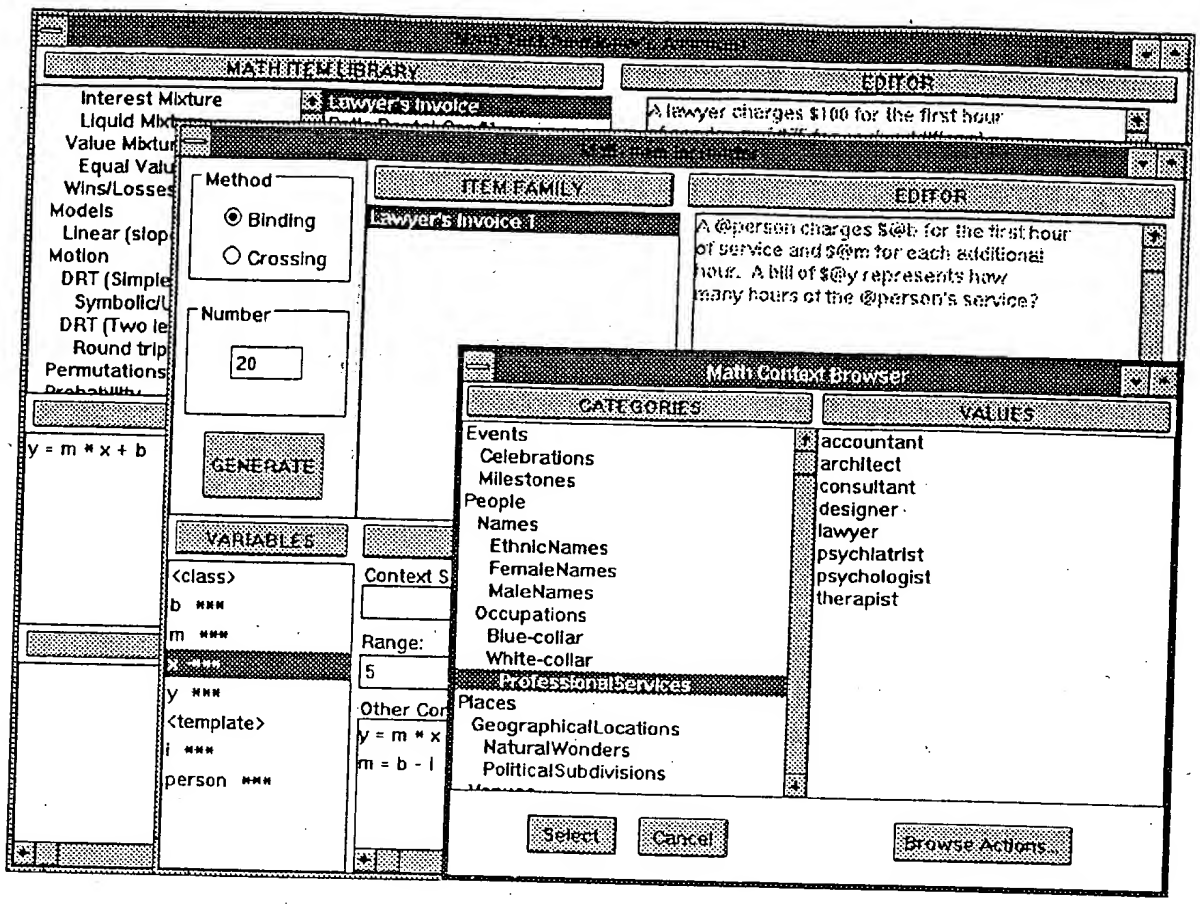


Figure 9. Interface of the Math Context Browser, showing selection of the Professional Services category.

difficulty. Another way to specify the values for a numeric variable is by writing new equations that derive the value of the current variable from the values of others.

As the user constrains the values of the variables by typing in values, making category selections, specifying ranges, or writing equations, the system automatically updates the list of variables that is displayed in the lower left corner of the Math Item Incubator. The display is updated to reflect which variables have been sufficiently constrained to allow for generation of variants (variables having this status are annotated with asterisks). When all the variables that appear in the template are annotated in this way, the system is prepared to generate variants. The user can specify the number of variants desired. The system does a breadth-first traversal of the space of all possible variants defined by the set of constraints; that is, it attempts to make the variants maximally dissimilar by changing the values of as many variables as possible between iterations.

Figure 10 shows a new set of 10 variants generated from the template in Figure 8. A variant involving an accountant is selected from the list. As shown, the values of all variables are consistent with the constraints specified in the template.

One important feature of the system is that it is quite easy to modify the templates themselves and thus greatly increase the range of variation associated with a particular parent item. In our current example, by switching information between the stem and the options, it would be quite easy to create a new template that would ask the examinee to determine the value of y (total bill) rather than the value of $x + 1$ (total hours worked). This amounts to changing the configuration of givens and goals across problems.

Prospects for Automatic Item Analysis

If items are characterized as instantiations of schemas, we seem to be in quite a good position to do automatic item analysis. Having this kind of handle on the deep structure of a problem may provide us with a host of important features that can be derived automatically and that may help in estimating difficulty. We have just begun to think about the implications of schema-based item authoring for automatic analysis, but here are some preliminary thoughts:

First, the schematic category from which an item is drawn may itself be a predictor of item difficulty (Marshall, 1995). So, for example, round-trip problems may on average be more difficult than two-legged DRT problems. One could imagine studying this at the rather molar level of the schemas themselves, or being more analytical and trying to predict the difficulty of individual schemas from their constituent equations. Using the latter approach, the round-trip and two-legged DRT schemas should operate very similarly because they share so many constituent equations.

Secondly, we might explore the possibility of predicting the difficulty of items based on the complexity of their search spaces (see Figure 3). Indeed, the Schema Compiler described earlier essentially provides us with a fledgling cognitive model that can

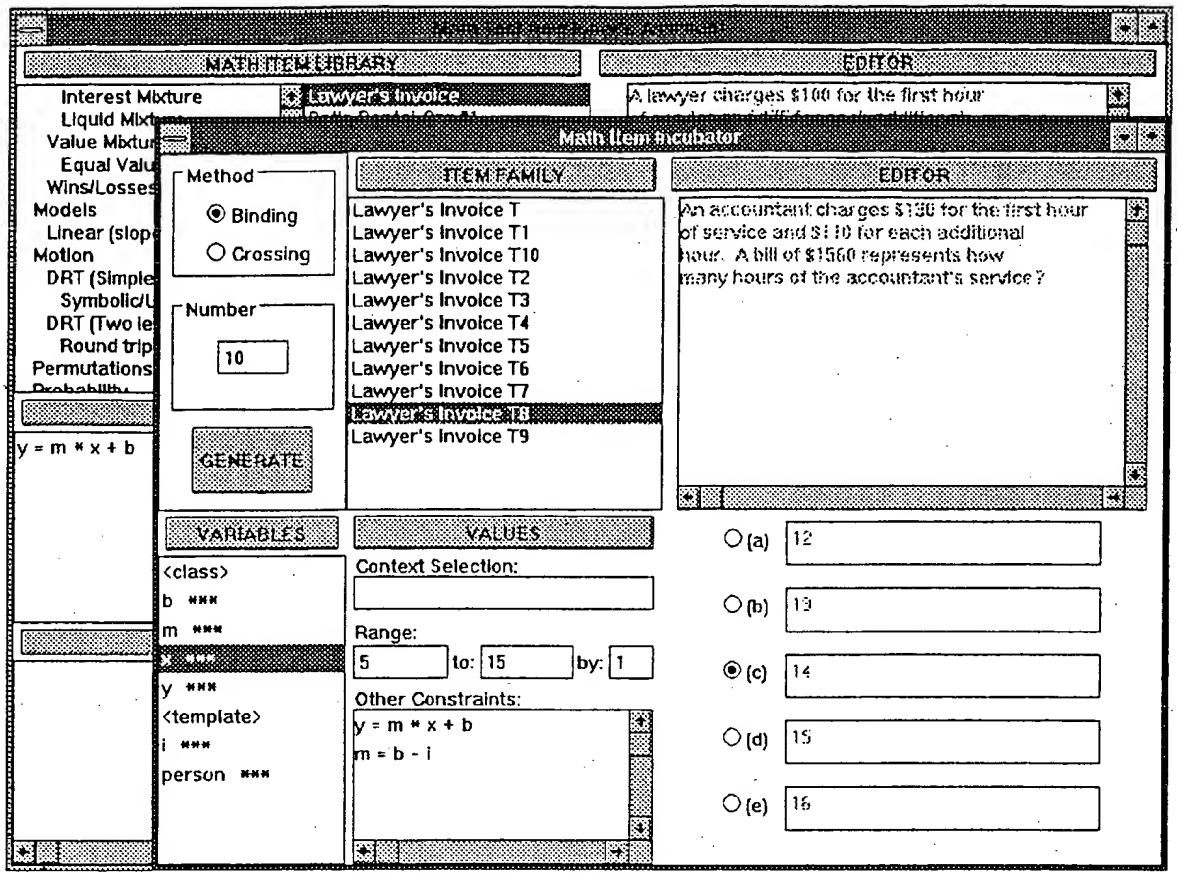


Figure 10. Ten new items generated from the template.

actually solve schema-based problems and characterize the complexity of the solution process. The search spaces derived by the Schema Compiler are the product of an exquisite interaction between the problem representation (the schema) and the problem-solving strategy (means-ends analysis). This interaction is such that two problems that involve the exact same set of four equations (i.e. have the same structural description) may have dramatically different search spaces.

III. Discussion

We have performed a system validation of the Mathematical Expression scoring routines, our first production automatic scoring application. The validation involved subjecting the prototype to one test and the operational ME scoring engine to three tests, one involving the analysis of 654 ME field trial responses, another with 6,834 field trial responses, and a third involving the analysis of 292 paraphrases of difficult expressions generated by GRE mathematics test development staff. In all cases, the routines did quite well. In the first case, the few disagreements between the scoring routines and human judges were due either to human error or to the presence of irrational expressions as responses. To deal with this latter problem, the ME interface has been reconfigured to disallow the entry of fractional exponents and other elements that render a response as irrational. For the second test, the production algorithm disagreed with human judgment in 26 of 6,834 cases because examinees were improperly entering subscripts and text strings into their expressions. Both entry problems, we believe, can be successfully addressed through the ME tutorial or minor processing modifications. The last test, which was intentionally designed to test the limits of the production algorithm, showed it to score 70% of the difficult paraphrases, rejecting the remainder due to the memory limitations of the minimally configured test center machines. Given the judicious choice of keys, we have no reason to expect these limitations to pose an operational problem, as relatively simple keys should keep the complexity of examinee responses within acceptable bounds.

Following the system validation of ME, we turned our attention to extensions involving the analysis of multiple-line responses and item authoring tools. We applied schema theory to the creation of two new analytical engines for multiple-line responses. The Schema Prover uses constraint logic programming techniques to explore the correctness and completeness of responses by attempting to "prove" the response from the schema specification, and vice versa. This scoring engine is optimized for the analysis of full responses involving the assignment of full- and partial-credit scores in assessment situations. The Schema Compiler uses means-ends analysis to generate a problem graph that represents all possible states of a schema-based problem and all paths from these states to goal states. This scoring engine is optimized for the analysis of partial or emerging solutions in instructional situations. The analyses in both cases depend critically on having the right representation of the schema. Although actually entering the schema is trivial, the analyst must strive to cast the schema at the finest level of granularity possible

and to include all possible quantities and relationships found in the problem situation. This results in the most extensive coverage of problem states and paths and minimizes the chance for "gaps" in the analysis.

Presently, both the Schema Prover and the Schema Compiler are limited to linear systems of equations, or, at least, systems that are linear once they are instantiated with known quantities. Future work will focus on extending our analytical power to non-linear equations and other kinds of elementary functions, for example, trigonometric and transcendental functions.

Although the Schema Compiler represents our first attempt at a cognitive model from which one could derive predictions of difficulty automatically, much work remains to be done. First, the schemas themselves need to be validated psychologically, as well as the strategy for searching them. The minimal means-ends analysis engine we are using now is almost certainly inadequate in that it is a purely working-backwards strategy that may be avoided by examinees because it imposes a severe working memory load (Sweller, Mawer, & Ward, 1983). We should explore the incorporation of some working-forward elements as well as elements of generate-and-test to better model the actual behavior of examinees. Protocol analyses of examinees solving problems would appear to be indispensable for informing this effort.

What are the implications of this work for the GRE Program? First, the Mathematical Expression automatic scoring engine appears to be functioning well from an accuracy perspective. At this stage, it is critical to test the routine's performance in a test-center-equivalent network setting with several individuals responding to ME items simultaneously. Although we believe performance will be adequate, we need to verify that the algorithm can score multiple ME responses in real time without dominating the network or timing out on what would otherwise be scorable entries.

Second, we will have to take into account earlier in the prototyping process test-center equipment constraints relevant to open-ended response formats requiring real-time scoring. This is, in fact, the approach we are pursuing with our current work on the Generating Examples response type (Bennett, Singley, Lee, & Morley, 1996), for which scoring routines are "precompiled." This approach allocates heavier preprocessing to ETS central-site machines and minimal processing to test-center equipment. By doing so, it sidesteps the limitations associated with the current approach. Among other things, GE does expression scoring, but using a very different approach from the symbol manipulation process employed by ME. (GE uses an "evaluation" method, plugging multiple values into the key and the response to see if the two expressions evaluate to the same result.) Although we are just beginning to evaluate the accuracy of this new approach, we are hopeful that it can supplement the current algorithm, perhaps by being called to score those difficult responses that ME rejects.

Third, we have in schema theory a general framework for integrating test creation, test preparation, and automatic analysis. The practical implication is that the same deep-

level item description (i.e., the item schema) can be used for classifying items, generating question variants, predicting difficulty, assembling tests based on a chosen range of content and item difficulty, delivering feedback as students solve problems, and scoring multiple-line responses. The basic machinery for implementing schema theory was completed as part of this project. In terms of efficiency, it is a substantial advance over the knowledge-based approach used in our earlier research and, we believe, has reached the point of practical utility for at least some initial applications. Unlike the earlier approach, with this one, large problem classes can be specified with relatively little work and with essentially no technical skill beyond mathematical content knowledge.

In our research, we have tried to lay the groundwork for a future generation of computer-based tests that broadens how and what we test. Schema theory offers exciting possibilities for doing that, including more cognitively based test designs, better automatic scoring methods, and feedback that can be used for test preparation or for new approaches to assessment (e.g., "dynamic" assessment). Considerable work remains to be done before these improvements are realized. We must still determine how broadly we can apply the schema framework in terms of content coverage, how well it will predict difficulty, how accurate it is in deciphering student responses, and how acceptable it is to mathematics test creation staff members.

References

Bennett, R. E. (1994). *An electronic infrastructure for a future generation of tests* (RR-94-61). Princeton, NJ: Educational Testing Service.

Bennett, R. E., & Sebrechts, M. M. (1996). The accuracy of expert-system diagnoses of mathematical problem solutions. *Applied Measurement in Education, 9,* 133-150.

Bennett, R. E., Sebrechts, M. M., & Rock, D. A. (1991). Expert-system scores for complex constructed-response quantitative items: A study of convergent validity. *Applied Psychological Measurement, 15,* 227-239.

Bennett, R., Singley, K., Lee, J. M., & Morley, M. (1996). *Generating Examples: A new response type for measuring quantitative reasoning.* (Proposal submitted to the Graduate Record Examinations Board.) Princeton, NJ: Educational Testing Service.

Marshall, S. (1995). *Schemas in problem solving.* New York: Cambridge University Press.

Mayer, R. E. (1981). Frequency norms and structural analysis of algebra story problems into families, categories, and templates. *Instructional Science, 10,* 135-175.

Norvig, P. (1993). *Paradigms of artificial intelligence programming.* San Mateo, CA: Morgan Kaufmann.

Sebrechts, M. M. (1992). From testing to training: Evaluating automated diagnosis in statistics and algebra. In C. Frasson, G. Gauthier, & G. I. McCalla (Eds.), *Intelligent Tutoring Systems* (ITS 1992 Proceedings) (pp. 560-566). Hiedelberg, Germany: Springer-Verlag.

Sebrechts, M. M., Bennett, R. E., & Katz, I. R. (1993). *A research platform for interactive performance assessment in graduate education* (RR-93-08). Princeton, NJ: Educational Testing Service.

Sebrechts, M. M., Bennett, R. E., & Rock, D. A. (1991). Agreement between expert system and human raters' scores on complex constructed-response quantitative items. *Journal of Applied Psychology,* **76**, 856-862.

Sheehan, K., & Mislevy, R. (1994). *A tree-based analysis of items from an assessment of basic mathematics skills* (RR-94-14). Princeton, NJ: Educational Testing Service.

Singley, M. K. (1995). Promoting transfer through model tracing. In A. McKeough, J. Lupart, & A. Marini (Eds.), *Teaching for transfer: Fostering generalization in learning*. Hillsdale, NJ: Erlbaum Associates.

Singley, M. K., Anderson, J. R., & Gevins, J. S. (1991). Promoting abstract strategies in algebra word problem solving. In *Proceedings of the International Conference on the Learning Sciences*. Charlottesville, VA: Association for the Advancement of Computing in Education.

Sweller, J., Mawer, R., & Ward, M. (1983). Development of expertise in mathematical problem solving. *Journal of Experimental Psychology: General, 112*, 639-661.

Tatsuoka, K., Birenbaum, M., Lewis, C., & Sheehan, K. (1993). *Proficiency scaling based on conditional probability functions for attributes* (RR-93-50-ONR). Princeton, NJ: Educational Testing Service.